Logic Algorithms for Natural Numbers

Frank M. Brown

TR - 108

July 1979

Abstract

Logic Algorithms for unary and binary, addition and multiplication on natural numbers are described. These algorithms may be viewed as recursive axioms of a first order logic. The correctness of the binary algorithms is proven relative to the unary algorithms and the time complexity of each algorithm is computed.

Contents

1. Introduction 1

2. Logic Algorithms for Natural Numbers 2

3. Correctness 4

4. Complexity 9

1. Introduction

One of mankinds greatest intellectual achievements was the development of positional numbering systems as opposed to unary numbers such as Roman numerals. In this paper in section 2 we axiomatize the operations of incrementation, addition and multiplication for both unary numbers and the simplest positional numbering system namely binary numbers. These axioms, as we point out may be viewed as algorithms which can be executed by the abstract machine also described in section 2. In section 3 we prove that the algorithms for binary numbers and the algorithms for unary numbers give essentially the same result. Finally in section 4 we calculate the time complexity of executing each of these algorithms on our abstract machine.

The purpose of all this is to illustrate a certain viewpoint or philosophy as to what computer science is all about, which put strongly (for emphasis) might be expressed as: "Its all just logic." To illustrate we note that the programming language is logic, the data on which the programs operate are terms of logic, the programs are axioms expressed in logic, the correctness (ie verification) conditions are also expressed in that same logic, and finally that the complexity equations are just other sentences of that logic.

Regardless of the truth or falsity of this viewpoint let us mention just in passing that if the automation of programming is a desirable goal, and if programming involves considerations about programs , their data, whether they are correct or not, and their complexity, then surely the ability to express all these things in a single language where everything interfaces to everything else, is itself a useful starting point. (For example notice how the complexity of addition calculation in section 4.5 depends on knowing that the number of bits in (Plus x y) is less than or equal to the number of bits in the larger of x or y plus 1. Thus rigorously a computer would have to define a length function in the programming language, then call the verifier to prove that

$$(\text{len}(\text{Plus } x\ y)) \leq (\max(\text{len } x)(\text{len } y)) + 1$$

and then use this fact back in the complexity proof.)

2. Logic Algorithms for Natural Numbers

Logic algorithms for unary and binary, increment, addition and multiplication are given below successively in sections 2.1, 2.2, 2.3, 2.4, 2.5 and 2.6. These algorithms consist of recursive axioms of the first order logic and may be executed as an algorithm by the following abstract machine. This machine consists of function definition expantion That is, given an axiom of the form

$$(\Psi\ x_1 \ldots x_n) = (\varphi\ x_1 \ldots x_n)$$

and an expression to be evaluated of the form

$$(\delta\ a_1 \ldots a_n)$$

if the structure of Ψ is identical to the structure of δ then the machine binds the variables $x_1 \ldots x_n$ respectively to the terms $a_1 \ldots a_n$ and replaces $(\delta\ a_1 \ldots a_n)$ by the new expressions to be evaluated: $(\varphi\ a_1 \ldots a_n)$.

It should be noted that this abstract machine is not so "abstract" as it might first appear, as it is very similar to the manner in which some practical contemporary computer languages work such as LISP and various automatic theorem provers for the first order logic.

2.1 Unary Increment

Suc1: (Suc a) = Sa

2.2 Unary Plus

P1: X + SY = S(X+Y)

P2: X + 0 = X

2.3 Unary Multiply

M1: X * SY = X * Y + X

M2: X * 0 = 0

2.4 Increment

One: (1 = S0)

Inc1: (Inc [X.0]) = [X.1]

Inc2: (Inc [X.1]) = [(Inc X).0)]

Inc3: (Inc nil) = [nil.1]

2.5 Plus

Plus1: (Plus [X.0] [Y.0]) = [(Plus X Y).0]

Plus2: (Plus [X.0] [Y.1]) = [(Plus X Y).1]

Plus3: (Plus [X.1] [Y.0]) = [(Plus X Y).1]

Plus4: (Plus [X.1] [Y.1]) = [(Inc (Plus X Y)).0]

Plus5: (Plus X nil) = X

Plus6: (Plus nil Y) = Y

2.6 Mult

Mult1: (Mult X[Y.0]) = [(Mult X Y).0]

Mult2: (Mult X[Y.1]) = (Plus [(Mult X Y).0] X)

Mult3: (Mult X nil) = nil

3. Correctness

We wish to show that the binary logic algorithms for incrementation, addition, and multiplication are correct. By "correct" we mean that each algorithm when viewed as axioms should have the same basic properties deducible from it as does some (hopefully simpler and easier to understand) standard algorithm when it too is viewed as axioms. Thus, in the case of natural numbers we accept the relatively simple axioms for unary natural numbers as the standard as to what properties natural numbers should have. We will therefore try to prove that the more complicated algorithms for binary natural numbers posess the same properties.

One approach to proving these properties would be to try to determine what properties held for unary natural numbers and then try to prove that each of those properties held for binary natural numbers. This approach has the disadvantages that each such property requires a separate proof and further that it is difficult to determine just what are all the relavent properties. For these reasons we feel that a different approach should be taken which is as follows: First define a 1-1 function on the two sets of axioms which will map elements of one domain into the other and then prove that this function is a homomorphism which preserves all the relevant properties. It will then follow in one fell swoop that all the relevant properties of unary natural numbers will hold for binary natural numbers.

We define a 1-1 function U from binary numbers onto unary numbers as follows:

Unary

U1: (U[X.b]) = 2 * (UX) + b

U2: (U nil) = 0

We prove that U is a homomorphism with respect to Inc, Plus, and Mult successively in sections 3.1, 3.2 and 3.3. These proofs are obtained by induction on the structure of binary numbers. The simple induction rule I1:

I1: $\forall X\ \varphi X \leftrightarrow (\varphi\,\mathrm{nil} \wedge (\forall X\ \varphi X \rightarrow \varphi[X.0] \wedge \varphi[X.1]))$

is used in sections 3.1 and 3.3. However, a course of values induction rule I2 on 2 variables is used in section 3.2:

$$\begin{aligned}\forall X \forall Y\ \varphi XY \leftrightarrow & \forall X\ \varphi X\ \mathrm{nil} \wedge \forall Y\ \varphi\ \mathrm{nil}\ Y \\ & \wedge \forall X \forall Y\ \varphi XY \rightarrow \varphi[X.0][Y.0] \wedge \varphi[X.0][Y.1] \wedge \varphi[X.1][Y.0] \wedge \varphi[X.1][Y.1]\end{aligned}$$

3.1 Correctness of binary increment

U(Inc X) = (Suc (UX)]: Suc1

U(Inc X) = 1 + UX : Induction rule 1

1. U(Inc nil) = 1 + Unil : Inc1

   U(nil.1) = 1 + Unil : U1

   2*Unil + 1 = 1 + Unil : U2

   2*0+1 = 1 + 0

   1 = 1

   ∎

2. U(Inc X) = 1 + UX → U Inc [X.0] = 1 + U[X.0]

   ∧ U Inc [X.1] = 1 + U[X.1]

   : Inc1,Inc2

   U(Inc X) = 1 + UX → U[X.1] = 1 + U[X.0]

   ∧ U[(Inc X).0] = 1 + U[X.1]

   : U1

   U(Inc X) = 1 + UX → 2*UX+1 = 1 + 2*UX

   ∧ 2*U(Inc X) + 0 = 1 + 2*UX + 1

   Hyp

   2*(1+UX) = 2*UX+1

   2*UX+2 = 2*UX+2

   ∎

3.2 Correctness of binary addition

U(Plus X Y) = UX + UY :Induction rule 2

1. U(Plus nil Y) = Unil + UY : PLUS6,U2

   (U Y) = 0 + UY

2. U(Plus Xnil) = UX + Unil :PLUS5,U2

   UX = UX + 0

3. U(Plus X Y) = UX + UY → U(Plus [X.0] [Y.0]) = U[X.0] + U[Y.0]

   $\wedge$ U(Plus [X.0] [Y.1])= U[X.0] + U[Y.1]

   $\wedge$ U(Plus [X.1] [Y.0])= U[X.1] + U[Y.0]

   $\wedge$ U(Plus [X.1] [Y.1])= U[X.1] + U[Y.1]

   : PLUS1,PLUS2,PLUS3,PLUS4,U1

   U(Plus X Y) = UX + UY →U[(Plus X Y).0] = 2*UX + 0 + 2*UY + 0

   $\wedge$ U[(Plus X Y).1] = 2*UX + 0 + 2*UY + 1

   $\wedge$ U[(Plus X Y].1] = 2*UX + 1 + 2*UY + 0

   $\wedge$ U[(Inc(Plus X Y)).0] = 2*UX + 1 + 2*UY + 1

   :U1

   U(Plus X Y) = UX + UY→2*U(Plus X Y) + 0 = 2*UX + 2*UY

   $\wedge$ 2*U(Plus X Y) + 1 = 2*UX + 2*UY + 1

   $\wedge$ 2*U(Plus X Y) + 1 = 2*UX + 2*UY + 1

   $\wedge$ 2*U(Inc(Plus X Y)) + 0 = 2*UX + 2*UY + 2

   :Hyp

   U(Plus X Y) = UX + UY→2*(UX + UY) = 2*(UX + UY)

   $\wedge$ 2*(UX + UY) + 1 = 2*(UX + UY) + 1

   $\wedge$ 2*(UX + UY) + 1 = 2*(UX + UY) + 1

   $\wedge$ 2*U Inc(Plus X Y) = 2*(UX + UY + 1)

$U(\text{Plus } X\ Y) = Ux + Uy \rightarrow \quad 2*U\ \text{Inc}(\text{Plus } X\ Y) = 2*(Ux + Uy + 1)$

$U(\text{Plus } X\ Y) = Ux + Uy \rightarrow U(\text{Inc}(\text{Plus } X\ Y)) = Ux + Uy + 1$ hyp

$U(\text{Inc}(\text{Plus } X\ Y)) = U(\text{Plus } X\ Y) + 1$

: generalize

$U(\text{Inc } Z) = UZ + 1$: Correctness of Inc

■

3.3 <u>Correctness of binary multiplication</u>

$U(\text{Mult } X\ Y) = UX * UY$: Induction Rule 1

1. $U(\text{Mult } X \text{ nil}) = UX * U\text{nil}$: Mult3

$U \text{ nil} = UX * U\text{nil}$

$0 = UX*0$

2. $U(\text{Mult } X\ Y) = UX * UY \rightarrow U(\text{Mult } X[Y.0]) = UX * U[Y.0]$

$\wedge\ U(\text{Mult } X[Y.1]) = UX * U[Y.1]$

: Mult1, Mult2

$U(\text{Mult } X\ Y) = UX * UY \rightarrow U[(\text{Mult } X\ Y).0] = UX * ((2*UY) + 0$

$\wedge\ U(\text{Plus}[(\text{Mult } X\ Y).0]X) = UX * (2*UY+1)$

:correctness of PLUS

$U(\text{Mult } X\ Y) = UX * UY \rightarrow U[(\text{Mult } X\ Y).0] = 2 * UX * UY$

$\wedge\ U[(\text{Mult } X\ Y).0] + UX = (2*UX*UY) + UX$

:U1

$U(\text{Mult } X\ Y) = UX * UY \rightarrow 2*U(\text{Mult } X\ Y) + 0 = 2 * UX * UY$

$\wedge\ 2*U(\text{Mult } X\ Y) + 0 + UX = (2*UX*UY) + UX$

Hyp

$2 * UX * UY = 2 * UX * UY$

$\wedge\ (2 * UX * UY) + UX = (2 * UX * UY) + UX$

■

4. Complexity

We are interested in determining the time it takes to execute these logic algorithms on the abstract machine defined in section 2. Each operation of that machine, consisting of a successful matching, binding and replacment step is considered to be of time complexity one. Unsuccessful matchings are counted as time complexity zero.

It should be noted that if these logic algorithms are rewritten as Horn clauses by unnesting the nested functions such as PLUS MULT and INC by use of the law: $\varphi(fx_1 \ldots x_n) \leftrightarrow (\forall z (fx_1 \ldots x_n) = z \rightarrow \varphi z)$ then if they are executed by a method similar to the operation described above modified so as to replace the matching step by a unification step then if each successful unification is counted as time complexity 1 and each unsucessful one is counted as 0 then the same time complexity will hold for both the resulting Horn clause algorithms and the original algorithms.

It may however be argued that basing complexity on the number of such operations (using a matching algorithms in our case, or a unification algorithm in the case of Horn clauses) is an unintuitive measure which has little relation to the real world. In particular, two critisisms come to mind: First, that our operation is not a fundamental operation in contemporary computers and thus defining all operations in terms of this one needlessly makes programs more complex than they need be. Our reply to this critisism is that our operation which consists of matching binding and replacement respectively implements the 3 basic machine operations of condition testing, assignment to a variable and program branching (which includes looping and recursion), and thus does consist of fundamental operations of a computer. A second critisism is that although our operation does consist of some fundamental computer operations, it does not contain all the fundamental computer operations. For example it does not contain arithmetic instructions for addition and multiplication. Our reply to this critisism is that if one really wishes one can indeed agree to let any call of the PLUS function or MULT function from themselves or each other have complexity 0 thus making all external calls to them have complexity 1. However, we also feel that addition and multiplication are not really all that funcamental at least for micro computers. For example the Motorola 6800 computer has no multiplication intruction and only a measly 8 bit addition instruction.

We give below in the table a summary of the complexity of the six functions described in section 2. This is followed successively in sections 4.1, 4.2, 4.3, 4.4, 4.5, 4.6 by derivations of the time complexity for each of these functions.

Time Complexity Summary

	Unary	Binary
increment	1	$\log X = n$
addition	X	$(\log X)^2 = n^2$
multiplication	X^2	$(\log X)^3 = n^3$

notes:

X is the magnitude of the number

n is the number of bits used to represent X as a binary number (re: 2^n=x).

4.1 Complexity of binary successor

The complexity equation is:

$C_s 1$: $C_s X = 1$

Thus the successor operations has constant complexity.

4.2 Complexity of unary addition

The complexity equations are:

C+1: ((C+ SY) = (C+ Y) + 1

C+2: (C+ 0) = 1

Their closed form solution is

$$C+Y = \sum_{K=0}^{Y} K = Y+1$$

Thus unary addition is linear with respect to the magnitude of the 2nd argument (ie hopefully the smallest argument).

4.3 Complexity of unary multiplication

The complexity equations are:

C*1: (C* X XY) = (C* XY) + (C+ X)

C*2: (C* X 0) = 1

Since C+ X equals X+1 we get:

C*'1 (C* XsY) = (C* XY) + (X+1)

C*'2 (C* X0) = 1

Their closed form solution is:

$$(C^* XY) = 1 + \sum_{K+1}^{Y} (X+1) = 1 + (X+1)Y = XY + Y + 1 = O(XY)$$

Thus is X and Y are the same size unary multiplication is quadradic $O(X^2)$ with respect to the magnitude of the numbers.

4.4 Complexity of binary increment

The complexity equations are:

$C_I^*1 \quad (C_I^*[X.0]) = 1$

$C_I^*2 \quad (C_I^*[X.1]) = C_I^* + 1$

$C_I^*3 \quad (C_I^*\ \text{nil}) = 1$

Rewritten in terms of the number of bits in each number such as [X.0]

these equations become:

That C_I^*X becomes $C_I^*2^n$ which is replaced by C_In

$C_I1' \quad (C_I\ n+1) = 1$

$C_I2' \quad (C_I\ n+1) = (C_I\ n) + 1$

$C_I3' \quad (C_I\ 0) = 1$

Since C_I1' and C_I2' are identical we can combine them:

$C_I1\text{-}2'' \quad (C_I\ n+1) = (C_I\ n) + 1$

$C_I3'' \quad (C_I\ \ 0) = 1$

The closed form solution for these equations is:

$(C_I\ n) = n+1$

which is easily proven by many induction on n.

$$\begin{tikzcd}
 & C_I\ n = n+1 \arrow[dl] \arrow[dr] & \\
C_I\ 0 = 1\ :C_I3'' & & C_I\ n = n+1 \rightarrow C_I\ n+1 = (n+1)+1 \quad :C_I1\text{-}2''
\end{tikzcd}$$

$1 = 1$

■

$C_I\ n = n+1 \rightarrow C_I\ n+1 = (n+1)+1$

$(n+1)+1 = (n+1)+1$

■

Thus the binary increment function is linear with respect to the number of bits in the number, and logarithemic (to base 2) with respect to the magnitude of the number:

$C^*_{I2}\, n = n+1$

and $C^*_I\, X = (\log_2 X)+1$

4.5 Compelxity of binary addition

The complexity equations are:

$C^*_p 1$: $(C^*_p\,[X.0][Y.0]) = (C^*_p\, XY)+1$

$C^*_p 2$: $(C^*_p [X.0][Y.1]) = (C^*_p\, XY)+1$

$C^*_p 3$: $(C^*_p [X.1][Y.0]) = (C^*_p\, XY)+1$

$C^*_p 4$: $(C^*_p [X.1][Y.1]) = (C^*_p\, XY)+1+C^*_I(\text{Plus } XY)$

$C^*_p 5$: $(C^*_p\, X\ \text{nil}) = 1$

$C^*_p 6$: $(C^*_p\ \text{nil}\ Y) = 1$

Rewritten in terms of the number of bits in each number such as [X.0] these equations become:

$C_p 1'$ $(Cp\ n+1\ n+1) = (C_p\ n\,m)+1$

$C_p 2'$ $(C_p\ n+1\ m+1) = (C_p\ n\,m)+1$

$C_p 3'$ $(C_p\ n+1\ m+1) = (C_p\ n\,m)+1$

$C_p 4'$ $(C_p\ n+1\ m+1) = (C_p\ n\,m)+1+C_I((\max\ n\,m)+1)$

$C_p 5$ $(C_p\ n\ 0) = 1$

$C_p 6'$ $(C_p\ 0\ m) = 1$

Note that if x has n bits and y has m bits then (Plus x y) has at most (max n+1 m+1) bits, which equals (max n m)+1 bits.

Since C_p1-C_p3 are identical these equations reduce to:

C_p1-3' $(C_p\ n+1\ m+1) = (C_p\ n\ m)+1$

C_p4' $(C_p\ n+1\ m+1) = (C_p\ n\ m)+1+C_I(\max\ n\ m)+1$

C_p5' $(C_p\ n\ 0) = 1$

C_p6' $(C_p\ 0\ m) = 1$

For a worst case complexity bound we can let n=m by extending the shorter number by a sequence of 0's. For example if the number of length n is [[[nil.1].1].1] and the number of length m is [nil.1] we can extend the number to be [[[nil.0].0].1]. Then since n=m the C_p complexity function needs only one arguement.

C_p1-3" $(C_p\ n+1) = C_p\ n)+1$

C_p4" $(C_p\ n+1) = C_p\ n)+1+(C_I\ n+1)$

C_p5-6" $(C_p\ 0) = 1$

By letting C_p calculate PLUS's complexity in terms of bits we lost the actual numbers that it worked on so we have no way of deciding the relative frequency as to when P1,P2,P3 is used and when P4 is used. Thus we do not know what weight to give to C_p1-3" and C_p4". However for a worst case analysis we can simply use C_p4" since it costs more than C_p1-3":

C_p1-4''' $(C_p\ n+1) \leq (C_p\ n)+1+C_I\ n+1$

C_p5-6''' $C_p\ 0 = 1$

Since $C_I\ n+1 = n+2$ we obtain:

C_p1-4'''' $C_p\ n+1 \leq (C_p\ n)+n+3$

C_p5-6'''' $C_p\ 0 = 1$

Furthermore, the closed form solution for C_p is:

$$C_p\ n \leq 1 + \sum_{K=1}^{n} (K+2) = 1 + \sum_{K=1}^{n} K + \sum_{K=1}^{n} 2 = 1 + \frac{n(n+1)}{2} + 2n$$

$$= \frac{n^2}{2} + \frac{n}{2} + 2n + 1 = \frac{n^2}{2} + \frac{5n}{2} + 1$$

Thus binary addition is quadradic $0(n^2)$ with respect to the number of bits, and hence is the quadradic of a logarithm with respect to the magnitude of the number:

$$C_p^*\ 2^n \leq \frac{n^2}{2} + \frac{5n}{2} + 1$$

$$C_p^*\ X \leq \frac{(\log_2 X)^2}{2} + \frac{5}{2}\log_2 X + 1 = 0((\log_2 X)^2)$$

4.6 <u>Complexity of binary multiplication</u>

The complexity equations are:

$C_m^*1 = (C_m^*\ X[Y.0]) = (C_m^*\ X\,Y)+1$

$C_m^*2 = (C_m^*\ X[Y.1]) = (C_m^*\ X\,Y)+1+(C_p^*[(\text{Mult}\ X\,Y).0]X)$

$C_m^*3 = (C_m^*\ X\ \text{nil}) = 1$

Rewritten in terms of the number of bit these equations become:

C_m1' $(C_m\ n\ m{+}1) = (C_m\ n\ m){+}1$

C_m2' $(C_m\ n\ m{+}1) = (C_m\ n\ m){+}1{+}(C_p\ (n{+}m){+}1\ n)$

C_m3' $(C_m\ n\ 0) = 1$

Note that if X has n bits and Y has m bits then (Mult X Y) has at most n+m bits:

$$(\#\text{bits in } X\cdot Y \leq \log_2(X\cdot Y) = \log_2(2^n\cdot 2^m) = \log_2 2^n + \log_2 2^m = n{+}m)$$

As in our calculation of the complexity of PLUS we can bound the complexity of the two argument $(C_p\ n\ m)$ function by letting n=m. Thus in the above equations we can replace $(C_p\ n{+}m{+}1\ n)$ by $(C_p\ n{+}m{+}1)$ getting

C_m1'' $(C_m\ n\ m{+}1) = (C_m\ n\ m){+}1$

C_m2'' $(C_m\ n\ m{+}1) = (C_m\ n\ m){+}1{+}(C_p\ n{+}m{+}1)$

C_m3'' $(C_m\ n\ 0) = 1$

Combining C_m1'' and C_m2'' for a worst case analysis we get:

$C_m1,2'''$ $(C_m\ n\ m{+}1) \leq (C_m\ n\ m){+}1\ (C_p\ n{+}m{+}1)$

C_m3''' $(C_m\ n\ 0) = 1$

Since $C_p\ n \leq \frac{n^2}{2} + \frac{5n}{2} + 1$ we get:

$$(C_m\ n\ m{+}1) \leq (C_m\ n\ m) + 1 + \frac{(n{+}m{+}1)^2}{2} + \frac{5(n{+}m{+}1)}{2} + 1$$

$$(C_m\ n\ 0) = 1$$

The closed form solution of $(C_m\ n\ m)$ is

$$C_m\ n\ m \leq 1 + \sum_{K=1}^{m} (1 + \frac{(n+(K-1)+1)^2}{2} + \frac{5(n+(k-1)+1)}{2} + 1)$$

$$= 1 + \sum_{K=1}^{m} \frac{(n+K)^2}{2} + \frac{5(n+K)}{2} + 2$$

$$= 1 + \sum_{K=1}^{m} (\frac{n^2+K^2+2n}{2} + \frac{5n+5K}{2} + 2)$$

$$= 1 + \sum_{K=1}^{m} (\frac{n^2+K^2+2Kn+5n+5K+4}{2})$$

$$= 1 + \sum_{K=1}^{m} (\frac{K^2}{2} + \frac{2n+5}{2} K + \frac{n^2+5n+4}{2})$$

$$= 1+\frac{1}{2} \sum_{K=1}^{m} K^2 + \frac{2n+5}{2} \sum_{K=1}^{m} K + \frac{n^2+5n+4}{2} \sum_{K=1}^{m} 1$$

$$= 1 + \frac{1}{2} (\frac{m^3}{3} + \frac{m^2}{2} + \frac{m}{6}) + \frac{2n+5}{2} (\frac{m^2}{2} + \frac{m}{2}) + \frac{n^2+5n+4}{2} m$$

$$= 1 + \frac{m^3}{6} + \frac{m^2}{4} + \frac{m}{12} + \frac{n}{2} m^2 + \frac{5}{4} m^2 + \frac{nm}{2} + \frac{5}{4} m + \frac{n^2}{2} m + \frac{5n}{2} m + 2m$$

$$= \frac{m^3}{6} + \frac{nm^2}{2} + \frac{n^2 m}{2} + \frac{3}{2} m^2 + \frac{3}{2} mn + (3\frac{1}{3})m + 1$$

Finally, assuming that m and n are initially the same size we find that the complexity of binary multiplication is cubic with respect to the number of bits.

$$C_{m\ n} \leq \frac{n^3}{6} + n^3 + 3n^2 + 3\frac{1}{3}n + 1 = \frac{7}{6}n^3 + 3n^2 + \frac{10}{3}n + 1 = O(n^3)$$

Therefore, the complexity of binary multiplication is the logarithm of the cubic with respect to the magnitude of the numbers being multiplied.

$$C_m^* 2^n = O(n^3)$$

$$C_m^* X = O((\log X)^3)$$